PUG Challenge Americas
2013 – Westford, MA

JSON 101
Data Access via AJAX in OE11

Presented by: Dustin Grau

BRAVEPOINT

About the Presenter

- Senior consultant at BravePoint, Inc.
- Began utilizing web technologies in 1996
  - HTML & JavaScript (referred to as DHTML)
- Developing apps with WebSpeed since 1999
- Working with RIA frameworks since 2008
- Lead UI architect for BravePoint's RIA projects
- Embraced HTML5 and WebApps in 2010
- Implementation expert for Application Evolution
- Year 3 at PCA - Thank You!

Agenda

- JSON Basics
- Vocabulary & Datatypes
- Using with AJAX
- Old Style Examples
- OpenEdge 11
- New Style Examples
- Q & A

JSON Basics

- JavaScript Object Notation
- Pronounced “jay-sun”
- Lightweight data-interchange format
- Similar to XML for describing data
- Less complex than XML
- Easy for humans and machines to parse
- Based on a subset of JavaScript language
- See json.org for more info

JSON Vocabulary

- Object: first node in a JSON structure
- Defined using curly braces: { and }
- Consists of name/value pairs separated with a comma
- JSON structures always start with an object

- Array: a list of values with an identical datatype
- Defined using square brackets: [and]
- Consists of single elements separated with a comma

JSON Datatypes

- JSON only knows of limited datatypes
  - Object: { name : value }
  - Array: [value, value, ...]
  - String: characters (quoted)
  - Number: integers, decimals
  - Boolean: true, false
  - Literal: null
- Everything else is made up of these types
  - Dates: string in ISO8601 format
  - Unicode: \u + 4 hex digits
  - Control: \b \f \n \r \t
  - Escape: \\ \/ \”

Do's and Don'ts

- DO: Use dot-notation to access values (object.property)
- DON'T: Use periods or spaces in property names
- DO: Use an underscore if you must have a space
- DON'T: Use symbols in property names, especially +/-
- DO: Use whole words in camelCase format for naming
- DON'T: Forget to end pairings with a comma (except last)
  - Remember the mantra: "name colon value comma"
- DO: Quote property names in data packets
  - { "myProperty": "myValue"}
- DON'T: Quote property names in JavaScript
  - { someProperty: "someValue" }

Representing Progress Data

Progress	JSON
CHARACTER LONGCHAR	String
INTEGER INT64 DECIMAL	Number
DATE DATETIME DATETIME-TZ	String (ISO8601)
LOGICAL	true false null
?	null

Representing Progress Data

Progress	JSON
TEMP-TABLE	Array of Objects
DATASET	Object with Arrays of Objects
RAW	String (Hex-Encoded)
CLOB	String (via LONGCHAR)
ROWID RECID	String (via CHARACTER)

Special Note: Dates

- Use DATE if you intend to only store a date
- Use INTEGER if you need to store the time
- Use DATETIME-TZ if you need an exact time
- JavaScript will use TZ offset information
- Avoid DATETIME due to TZ ambiguity!

DATE	01/01/2013	2013-01-01
DATETIME	01/01/2013 16:20:10.300	2013-01-01T16:20:10.300Z
DATETIME-TZ	01/01/2013 16:20:10.300-04:00	2013-01-01T16:20:10.300Z-04:00

Special Note: Extents & Symbols

- Fields with an EXTENT will be converted to an array
- Applies to all datatypes that allow for EXTENT
- This is perfectly legal syntax for JSON, however...
  - This is not a normalized form of data
  - Typically there is no direct UI representation
  - May be difficult to parse/use on the JS side

- Some symbols indicate math operations (+/-)
- JS doesn't care about spaces between operators
- Confuse JS into thinking it should concatenate strings
- Breaks convention of dot-notation access for properties

Sample Structure

```
{
      “myString”: “some character value”,
      “myNumber”: 3.1415,
      “isTrue”: true,
      “hasNull”: null,
      “myDate”: “2013-03-25T00:50:00Z-04:00”,
      “ttCustomer”: [
            {“CustNum”: 1, “CustName”: “Target”},
            {“CustNum”: 2, “CustName”: “Wal-Mart”}
      ]
}
```

AJAX Considerations

- Still using XmlHttpRequest Object
- Can receive data as both XML and plain text
- JSON should always be in plain text format
- Modern browsers have a window.JSON object
  - JSON.parse() to convert to object
  - JSON.stringify() to convert to text
  - Desktop: FF 3.5+, Safari 4+, Chrome 4+, IE 8+
  - Mobile: iOS 4+, Android 2.1+, BB 7+
- Stringify'd JSON is just properly escaped
- KISS: Stick to the datatypes, don't pre-format!
  - Let the UI do the formatting
  - Send enough data for the UI

Old Style Examples

- Simple AJAX/JSON retrieval
- Traditional AJAX using CGI wrapper
- Sending simple parameters
- Accessing data client-side
  - Simple objects/properties
  - Complex objects (e.g. tables)

Problems

- Output limited to simple strings or tables/datasets
- Parameters can only be passed on the URL
- Error handling is defined on a per-file basis
  - Not easily reusable (at this time)
  - Not inclusive of all possible errors
- Lots of overhead for headers, content formatting

- WEB-CONTEXT:IS-JSON detects JSON requests
- Based on Progress.Json.ObjectModel class
- JsonConstruct is an abstract class for objects/arrays
- The ObjectModelParser class
  - Parse() method accepts WEB-CONTEXT:HANDLE
  - Returns an object or array as JsonConstruct
    - Must cast into either JsonObject or JsonArray
    - Remember, structures must begin as an object!
- Errors thrown as Progress.Json.JsonParserError
- Use getClass() to find class type of a JsonConstruct

OE11 JsonObject

- Check for presence of properties via Has()
- Get datatype of property via GetType() as integer
  - Use JsonDataType to determine from integer
  - String, Number, Boolean, Object, Array, Null
  - Corresponds to 1, 2, 3, 4, 5, 6 (respectively)
- Use specific method to return actual value
  - Get[type]() where type is standard ABL datatype
  - Also supports GetJsonObject() and GetJsonArray()
  - See documentation for JsonObject

OE11 JsonArray

- Indexing in a JsonArray is 1-based
- All elements should be of identical datatype
- Determine size of the array via LENGTH property
- Use specific method to return actual value
  - Get[type]() where type is standard ABL datatype
  - Also supports GetJsonObject() and GetJsonArray()
  - See documentation for JsonArray

Sample Parser Code

```
USING Progress.Json.ObjectModel.*.

DEFINE VARIABLE jsonParser  AS ObjectModelParser NO-UNDO.
DEFINE VARIABLE jsonRequest AS JsonObject        NO-UNDO.

IF WEB-CONTEXT:IS-JSON THEN DO:
  jsonParser = NEW ObjectModelParser().
  jsonRequest = CAST(jsonParser:parse(WEB-CONTEXT:HANDLE,
                                      JsonObject)).

  CATCH err AS Progress.Json.JsonParserError:
    MESSAGE SUBSTITUTE("Error at byte &1", err:offset).
  END CATCH.
  FINALLY:
    DELETE OBJECT jsonParser NO-ERROR.
  END FINALLY.
END.
```

New Style Examples

- Introducing the JSON Parser
- Accessing specific elements
- Creating a simple AJAX gateway
  - Packaging for JSON-RPC

Not Covered

- Drilling into nested structures, ala XPath style
  - Example: object.array[i].property
  - { ttExample: [{ CustNum: 2 }] }
  - Uses covered principles in a recursive manner
- Better error handling and HTTP status output
  - e.g. 404 = Method Not Found
- Better handling of parameters for dynamic calls
  - Detect type of parameter and convert accordingly
  - Proper error handling for missing parameters
  - Dealing with table and dataset handles

Summary

- JSON is an easy to use format for data exchange
- Has a simple vocabulary and syntax (smaller than XML)
- Readily supported by most JS frameworks and tools
- Better than plain text, but just as easy to construct
- OE11 provides a simple means of parsing JSON input
- Arbitrary JSON data can be easily nested for output
- Be aware of datatype conversions with OE
- Can be extended to JSON-RPC for advanced concepts
- Need to plan for runtime errors, and how to handle them
- Create a common Class to do parsing of parameters
- JSON-RPC + OE11 + CALL = Simple RPC Gateway

Thank You!

Questions?

dgrau@bravepoint.com

http://apevolution.com